Vol.2013-GN-86 No.38
Vol.2013-CDS-6 No.38
2013/1/17

低消費電力化のための実行タスクの動的なプロセッサリソース割り当て機構

榊原 宏章[†1]　白石 善明[†1]　岩田 彰[†1]

専門家による遠隔診断等の動画像通信などのように，汎用携帯端末を長時間利用できるように低消費電力化の需要が高まっている．汎用携帯端末には多種のプロセッサが搭載されてきており，消費電力当りの処理能力向上にはヘテロジニアスコンピューティング技術の利用が考えられる．しかし，ハイパフォーマンスコンピューティング分野で行われている負荷分散や自動調節等の研究成果は互換性の点から汎用端末にそのまま適用できない．本稿では，汎用携帯端末においてプロセッサ抽象化 API 利用アプリケーションが汎用的に動作可能な方式を提案する．提案方式を Linux OS 上で OpenCL を用いて実装した．提案方式は汎用的に利用可能であることと動画像通信を例にして省電力効果があることを確認した．

A Dynamic Processor Resources Allocation Mechanism for Energy Saving

HIROAKI SAKAKIBARA[†1]　YOSHIAKI SHIRAISHI[†1]
AKIRA IWATA[†1]

There is a growing demand for a long time use by low power consumption to be able to use general-purpose mobile devices. Since the various processors have been mounted on a general-purpose mobile device, using heterogeneous computing technology to improve processing ability per power consumption can be applied. Although the research results about load balancing or automatic adjustment have been shown in the field of high performance computing, they are not directly applicable to general-purpose device in terms of compatibility. This paper proposes a method that an application can use API abstract processor in general-purpose mobile devices. The implementation of the proposed method using OpenCL on Linux OS is shown. We confirm that the proposed method is generically available and it gives an energy saving effect in an example of video communication.

1. はじめに

高性能な携帯端末の普及に伴い，携帯端末は情報の閲覧や通信だけでなく財布の代替や個人認証トークンとしても利用されており，携帯端末を長時間利用する需要は大きい．他にも，医療や機器修理の専門家による遠隔診断のように，迅速かつ継続的な遠隔での確認及び操作に携帯端末が使われることが増えてきている．このような動画像通信において携帯端末での低消費電力化の需要は大きい．また，そのような用途においては特定の携帯端末に利用を制限するのではなく，アプリケーション利用者が所持している各自の汎用携帯端末を用いて利用可能であることが望ましい．

既に汎用携帯端末には複数の高性能プロセッサが搭載されているため，消費電力当りの処理能力向上にはヘテロジニアスコンピューティング技術の利用が考えられる．ヘテロジニアスコンピューティングは，複数のプロセッサを活用することで高速化・省電力化を図る技術である．CPU の単一コア性能の向上限界に際し，CPU のマルチコア化が進み，またベクトル演算専用に設計された GPU の汎用目的利用である GPGPU（General Purpose GPU）が考えられた．GPGPU は大量のデータを処理する場合に向いており，計算の種類によっては高いパフォーマンスを発揮できるようになった．

GPU を始め，CPU やその他のアクセラレータ等，様々なプロセッサにおいて並列処理を行うことができ，かつヘテロジニアスな構成が一般的になったにも関わらず，対応製品や開発環境等が各ベンダに依存しているという問題があった．そこで OpenCL[1]というヘテロジニアスな並列コンピューティング用フレームワークが標準団体の Khronos Group によって策定され，対応が進んでいる．

ヘテロジニアスコンピューティングに関する研究は特にハイパフォーマンスコンピューティング（HPC）の分野において盛んに行われており，スーパーコンピュータ等に利用されている[2]．しかし，HPC 分野で行われている既存研究を汎用端末に適用することは，プログラムの改変や端末毎のパフォーマンス調節といった互換性の面から困難である．

PACUE[3]は，API フックを用いてプロセッサの切り替えを行う方式であり，プロセッサの割り当てを決定する機能の分離を行うことで互換性の課題を解決している．しかし，プロセッサ抽象化 API を用いる際には使用プロセッサだけでなく，タスクに割り当てるリソースの調節が処理時間に影響するが，それは端末や利用状況によって異なる．多様な環境が想定される汎用端末においては，このリソースの

†1 名古屋工業大学
Nagoya Institute of Technology

ⓒ 2013 Information Processing Society of Japan

調節も必要である．

また，前述したように，汎用携帯端末における低消費電力な高速処理の需要は大きく，プロセッサの性能向上等の環境が整いつつある．加えて，一般に汎用携帯端末で用いられる SoC（System on Chip）における，ダークシリコン問題[4]と呼ばれるマルチコア CPU 性能向上に関して，GPU の占有面積の大きさ等からも汎用携帯端末でのヘテロジニアスコンピューティングへの期待は大きい．つまり，汎用携帯端末においてプロセッサ抽象化 API を利用する汎用的なアプリケーションを開発・実行できることが望ましい．

本稿では PACUE の手法を拡張して，動的なリソース調節機能を追加し，汎用携帯端末においてプロセッサ抽象化 API 利用アプリケーションが動作可能な方式を提案する．提案方式を Linux OS 上に OpenCL を用いて実装し，汎用的な利用可能性と動画像通信を例にとした効果について確認する．

2. 関連研究と低消費電力化に向けた要件

動的に使用プロセッサ切り替えを行う機構に関する研究として，消費電力当りのパフォーマンスを最大化するプロセッサを選択するランタイム環境を構築した研究[5]や実行速度の変化を計測して動的な最適化を行う研究[6]等がある．また，ヘテロジニアス環境での最適化のためのアルゴリズムに関する研究には，CPU・GPU 間におけるタスクスケジューリングアルゴリズムを提案した研究[7]や最適化効果を定量的に予測するための実行時間予測モデルを作成した研究[8]等があり，動的なプロセッサの変更が消費電力当りのパフォーマンス向上に効果的であることを示している．

これらの研究の多くは HPC 向けに考えられており，汎用端末に適用することが難しい．HPC では多くが特定のコンピュータの構成に合わせてカスタマイズしたプログラムが動作する環境を想定している．それに対し，汎用端末では多様なコンピュータの構成において様々なアプリケーションが実行される環境を想定する必要があり，かつ HPC に比べプログラムの改変が困難である．また，汎用端末での普及には既存プログラムに対する互換性が必要である．そのような互換性に関する課題の解決を目指した研究として PACUE[3]がある．

PACUE はバイナリを改変せずに動的なプロセッサの変更と，利用状況に応じた使用プロセッサの選択を可能とする方式である．PACUE の基本的なアイデアはプロセッサの切り替えを行う機構と割り当てを決定する機構を分離することによって，既存アプリケーションへの互換性と切り替え決定方法に対する柔軟性を得る点にある．プロセッサ抽象化 API を呼び出す際に API フックを用いることによって，バイナリの改変を不要としながら動的プロセッサ変更による負荷分散を実現している．また，端末情報監視を，割り当てを決定する機構で行うことによって，利用状況に合った使用プロセッサの選択を行うことができる．

しかし，プロセッサ抽象化 API を用いる際には，タスクに割り当てるリソースの調節がパフォーマンスに影響する．調節にはハードウェアアーキテクチャを意識する必要があり，例えば，同時に並列できるスレッド数に合わせて調節を行う必要がある．つまり，処理時間を最小化するタスクに割り当てるリソースは使用プロセッサ毎に異なり，また，利用状況によっても異なる．割り当てるリソースによってはプログラムが実行できないこともあるため，多様な実行環境が想定される汎用端末で動作するアプリケーションにおいてプロセッサ抽象化 API を利用するにはこの調節も必要である．

汎用端末において低消費電力な高速処理を実現するためには，以下の要件があげられる．

[負荷分散] 特定プロセッサへの処理の集中を回避

[自動リソース調節] 端末構成や利用状況に適したリソース配分

[互換性] コードの変更なしで方式を利用可能

3. 提案方式

本研究は，汎用携帯端末における低消費電力な高速処理の実現のために，2 章で述べた要件を満たす，動的なプロセッサリソースの割り当て方式を提案する．

提案方式の概要を図 1 に示す．この図は PACUE の考え方を継承した構成であり，PACUE における“プロセッサ指定変更機構”が“プロセッサリソース割り当て機構”，“利用状況取得機構”が“配分機構”にあたる．

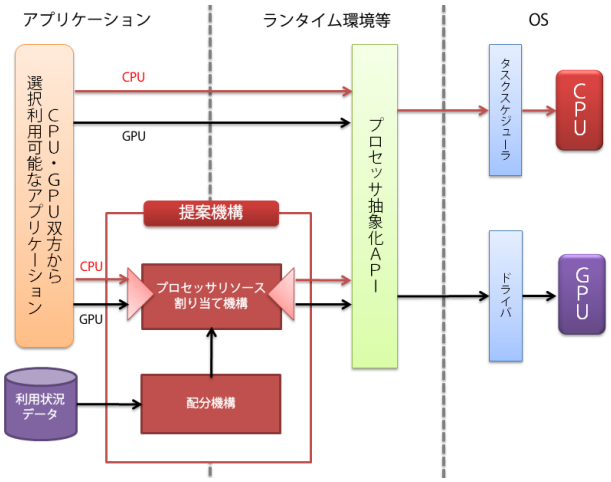

図 1　提案方式の概要

2 章で述べたように，PACUE の基本的なアイデアはプロセッサの切り替えを行う機構と割り当てを決定する機構を分離することによって，既存アプリケーションへの互換性と切り替え決定方法に対する柔軟性を得る点にある．

提案方式はこのアイデアを継承した構成をとる．これにより，PACUE が持つ互換性の要件を満たす．また，PACUE と同様にプロセッサの動的な選択を可能とすることで，負荷分散の要件を満たす．

提案方式ではそれに加え，自動リソース調節の要件を満たすように，タスクへ割り当てるリソースの動的な調節を行う．自動リソース調節により，多様な汎用携帯端末において個々の端末性能に応じたパフォーマンスを得ることが期待できる．また，端末に応じた最適化を外部に任せることは，アプリケーション開発コストを下げることにもつながる．

ワットパフォーマンスの高い GPU の活用や負荷分散，自動リソース調節によって安定した高速化を図ることができれば消費電力量の削減につながる．これは特に継続的に一定量の処理を行う動画像通信のような場合では，一回の処理の高速化による省電力効果は大きく，より長時間のアプリケーション実行を可能にすると期待できる．

提案方式の目的は，端末毎の差異を意識することなく，消費電力が少なくて，アプリケーションを高速に実行できる環境を構築することである．ただし，そのためには端末情報や過去の実行結果等からプロセッサ及びリソースの配分を決定するアルゴリズムが必要となるが，本稿の対象とはしていない．

4. 設計

4.1 想定環境

提案方式の想定環境は，マルチコア CPU と 1 つ以上の GPGPU に対応した GPU が搭載された汎用端末とする．

想定するアプリケーションは，CPU と GPU の双方からプロセッサ抽象化 API を通じて実行可能なアプリケーションとする．ただし，どちらか一方のみ対応可能なアプリケーションでも，指定されたプロセッサを利用して動作を行えるものとする．

4.2 構成

提案方式の全体構成を図 2 に示す．提案方式は大きくプロセッサリソース割り当て機構と配分機構の 2 つの機構を持ち，各々が複数個のモジュールを持つ．

ここでは各機構について概要を説明し，各モジュールの動作については次項において説明する．

(1) プロセッサリソース割り当て機構

プロセッサリソース割り当て機構は，アプリケーションとプロセッサ抽象化 API の間に介入して，プロセッサやリソースの割り当てを動的に変更する．変更の際には配分機構に問い合わせを行い，使用すべきプロセッサ，割り当て

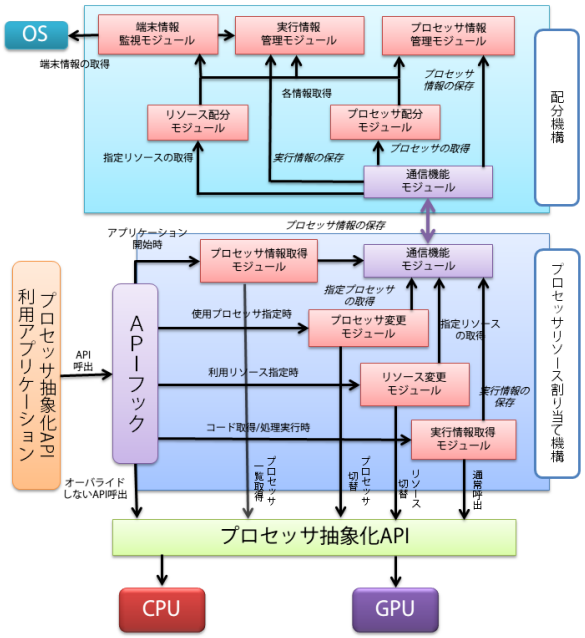

図 2 提案方式の構成

るリソースを受け取る．介入には API フックを用いることで，アプリケーションやプロセッサ抽象化機構の改変を行わずに動作することができる．

また，利用可能プロセッサ一覧等の端末情報や実行時間等の実行情報の取得を行い，配分機構へ渡す．

(2) 配分機構

配分機構は，プロセッサリソース割り当て機構からの問い合わせに対し，使用すべきプロセッサ，割り当てるべきリソースを返す．

また，配分アルゴリズムに必要となるプロセッサ使用率等の端末情報及び実行時間等の実行情報の取得・保存機能を持つ．

これら 2 つの機構は分離しているため，両者の間で問い合わせや値の取得等を行うためには通信が必要となる．そのため 2 つの機構は共に通信機能モジュールを持つ．

4.3 機能とその動作

(1) 使用プロセッサの指定

アプリケーションが処理に使用するプロセッサを指定した際に，アプリケーションが指定した以外のプロセッサが適切であると判断された場合，使用プロセッサの指定を変更する．

プロセッサリソース割り当て機構は API フックによって介入し，プロセッサ変更モジュールを呼び出す．プロセッサ割り当てモジュールは配分機構に問い合わせを行い，プロセッサ配分モジュールを呼び出す．プロセッサモジュールは端末情報や実行情報等から使用すべきプロセッサを決定し，プロセッサリソース割り当て機構のプロセッサ変更モジュールへ返す．プロセッサ変更モジュールは受け取っ

ⓒ 2013 Information Processing Society of Japan

た使用するプロセッサに変更して介入したプロセッサ抽象化 API を呼び出す.

(2) **割り当てリソースの指定**

アプリケーションが処理に割り当てるリソースを指定した際に，アプリケーションが指定した以外のリソースが適切であると判断された場合，割り当てリソースの指定を変更する.

プロセッサリソース割り当て機構は API フックによって介入し，リソース変更モジュールを呼び出す．リソース変更モジュールは配分機構に問い合わせを行い，リソース配分モジュールを呼び出す．リソース配分モジュールは端末情報や実行情報等から指定すべきリソースを決定し，プロセッサリソース割り当て機構のリソース変更モジュールへ返す．リソース変更モジュールは受け取った指定すべきリソースに変更して介入したプロセッサ抽象化 API を呼び出す.

(3) **実行情報の取得・保存**

プロセッサ及びリソースの配分を決定するために必要となる実行時間や実行プログラム情報等の実行情報の取得・保存を行う.

プロセッサリソース割り当て機構は API フックによって介入し，実行情報取得モジュールを呼び出す．実行情報取得モジュールは値を保持するか，または配分機構に送る．処理の終了時に今回の実行時に指定されたプロセッサ，リソース等と共に実行情報管理モジュールによって保存する.

(4) **端末情報の取得・保存**

プロセッサ及びリソースの配分を決定するために必要となるプロセッサ使用率等の端末情報の取得・保存を行う．取得する情報によって動作が異なる.

プロセッサ抽象化 API から取得する情報の場合，プロセッサリソース割り当て機構は API フックによって介入し，端末情報取得モジュールを呼び出す．端末情報取得モジュールはプロセッサ抽象化 API を使用して端末情報を取得し，プロセッサ情報管理モジュールによって保存する.

OS から取得する情報の場合，配分機構は定期的に端末情報管理モジュールを呼出し，OS の端末情報を取得し，端末情報管理モジュールによって保存する.

5. 実装

5.1 実装環境

OS には Ubuntu 10.04 LTS 32bit を用いる. API には CPU・GPU を利用することのできるコンピューティングフレームワークである OpenCL を使用することとした．OpenCL の実行環境及びコンパイラには CPU は AMD APP SDK 2.7 With OpenCL 1.2 Support[9]，GPU は NVIDIA の NVIDIA CUDA 4.2[10]を利用した.

なお，OpenCL では演算を行うプロセッサをデバイスと呼称するため，以降プロセッサをデバイスと呼ぶ.

表 1 実装環境

OS	Ubuntu 10.04 LTS 32bit
API	OpenCL
実行環境及びコンパイラ	[CPU] AMD APP SDK 2.7 With OpenCL 1.2 Support [GPU] NVIDIA CUDA 4.2

5.2 実装の方針

(1) **API フック**

Linux 系 OS で API フックを行う方法には LD_PRELOAD や LKM 等幾つかの方法があるが，本研究ではフックの確実性や実装及び利用の容易さから，ラッパーDLL の方法を用いた.

OpenCL 利用アプリケーションはラッパーDLL のオーバライドされた API を呼出し，ラッパーDLL からオリジナルの DLL の API を呼び出すことで動作する.

(2) **常駐アプリケーション**

常駐アプリケーションはラッパーDLL からの問い合わせに対し,使用するデバイスやリソース量等の指示を返す．また，端末情報を取得・保存，実行情報の保存等を行う.

(3) **機構間通信**

2 つの機構はプロセス間通信によって，使用するデバイスやリソース量の問い合わせ，値の取得等を行う．Linux 系 OS でのプロセス間通信の方法は INET ドメインソケット通信や共有メモリ等幾つかの方法があるが，INET ドメインソケットよりも高速な UNIX ドメインソケット通信を用いた.

(4) **並列プログラミングモデル**

並列プログラミングモデルにはデータ並列とタスク並列があり，OpenCL には両方の API が用意されている．本研究は CPU･GPU 共に実行可能であるような処理を想定としている．GPU において，1 つのデバイスに異なる処理を並列実行させるという意味でのタスク並列を行うことは標準では困難であるため，データ並列について考える.

5.3 具体的な動作

提案方式のモジュール構成を図 3 に示す.

OpenCL は，プラットフォームを一意に表すプラットフォーム ID（cl_platform_id），デバイスを一意に表すデバイス ID（cl_device_id），デバイスの種類を表すデバイスタイプ（cl_device_type）を持つ．OpenCL の処理は，まず仮想的な実行環境として 1 つ以上のデバイスに関連付けられたコンテキスト（cl_context）を生成し，これを利用してコマンドキュー（cl_command_queue）を生成する．メモリ転送や計算等のタスクは，このコマンドキューを介して実行する．OpenCL 利用アプリケーションが行ったこれらの API 呼出を Wrapped OpenCL API によってフックし，値を変更したのちにオリジナルの OpenCL DLL の API を呼び出すこ

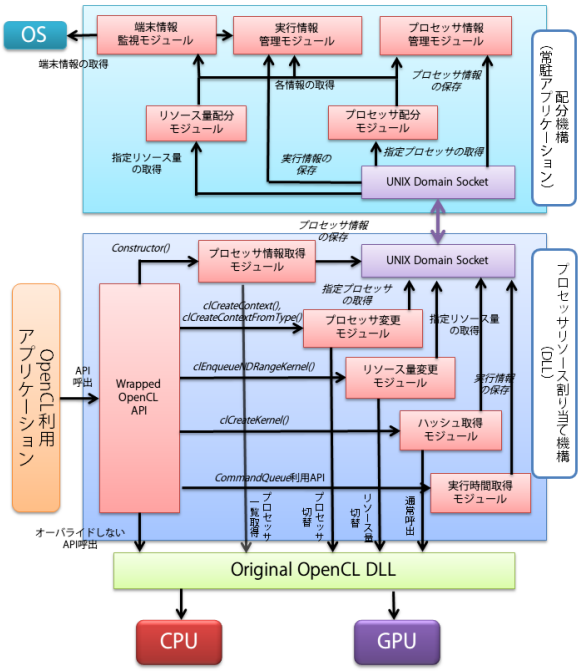

図 3　提案方式のモジュール構成

とによって，プロセッサリソースの動的な割り当てを行う．

OpenCL 利用アプリケーションがタスクにプロセッサを割り当てようとした際，Wrapped OpenCL API はプロセッサ変更モジュールを呼出し，デバイス ID 等の値を変更することで，タスクに割り当てるプロセッサの動的な変更を行う．

また，OpenCL のデータ並列プログラミングモデルにおいては，各演算ユニットにおいて別々のデータを処理させるため，インデックス空間を持つ．各処理の単位をワークアイテムと呼び，ワークアイテム ID によって識別する．また，ワークアイテムの集合の単位をワークグループと呼び，ワークグループ ID によって識別する．処理に使用するワークアイテムの総数をグローバルアイテム数，ワークグループ当りのワークアイテムの数をローカルアイテム数と呼ぶ．本実装で利用する実行環境において，ワークアイテムとワークグループは，GPU では演算コア，演算ユニットに対応し，CPU では軽量スレッド，OS スレッドに対応する．OpenC 利用アプリケーションがタスクにリソース量を割り当てようとした際，Wrapped OpenCL API はリソース量変更モジュールを呼出し，ワークアイテムの値を変更することで，タスクに割り当てるリソース量の変更を行う．

5.3.1 使用デバイスの指定

4.3(1)の機能に対応し，API で指定するデバイス ID を変更することにより，使用デバイスの変更を行う．

OpenCL コンテキストの生成方法には二種類の API が用意されており，clCreateContext()の場合はデバイス ID を指定，clCreateContextFromType()の場合はデバイスタイプとプラットフォーム ID を指定する必要がある．また，コマンドキューを生成する際には生成したコンテキストとデバイス ID を指定する必要がある．

PACUE では CPU と GPU を利用するのに 1 つのフレームワーク実装を用いていたため，ICD(Installable Client Driver)の機能を用いて，コンテキストを生成する際に使用可能なデバイスを全て指定しておき，コマンドキューを生成する際に使用するデバイス ID を指定することで変更を行う．そのためコマンドキューclCreateCommandQueue()がデバイス ID を変更することのできる最後のタイミングとしている．

しかし，異なるフレームワーク実装を同時に利用した場合，実装毎に互換性がないため，複数のプラットフォームが利用可能なコンテキストを生成することはできない．そこで本実装では CPU と GPU で異なるフレームワーク実装を利用しているため，コンテキストを生成する際に使用可能なデバイスを全て指定することはできない．そのため本実装ではコンテキストの生成時がデバイス ID を変更することのできる最後のタイミングとする．ただし，多様なデバイスを搭載している可能性の高い汎用端末では，異なるフレームワーク実装を利用する状況は頻繁に起こり得ると考えられる．

本実装では，コンテキストを生成する API を呼ぶ際に介入し，変更するプラットフォーム ID やデバイス ID，デバイスタイプを取得し，値を変更する．

5.3.2 使用ワークアイテム数の指定

4.3(2)の機能に対応し，データ並列処理実行時に API で指定するワークアイテム数を変更することにより，タスクに割り当てるリソース量の変更を行う．

ただし，アプリケーションによって変更しては正常に動作しない場合がある．例えばグローバルワークアイテム ID を利用するようなカーネルコードの場合，グローバルアイテム数の変更を行っては正しい処理が行われない．その判別をするためにはコード解析等プログラムの情報が必要である．本実装ではその判断を行わないが，幾つかのフレームワーク実装で，null の指定を行うことでワークアイテム数の自動調節を行うという機能が提供されている．そのため，本実装でも null が指定されていた場合に対応可能とした．

データ並列処理の実行 API は clEnqueueNDRangeKernel()であり，この API に介入し，処理に割り当てるリソース量の変更を行う．リソース量の変更はカーネル関数毎に行う．

5.3.3 実行情報の取得・保存

4.3(3)の機能に対応し，デバイス及びリソースの配分決定の補助とするため，本実装ではカーネルコードハッシュの取得と，実行時間の取得を行う．

(1)　カーネルコードハッシュの取得

実行情報の同定等に必要と考えられるため，デバイスで実行するカーネルコードのハッシュを取得する．ハッシュ

のみを送信するのは，アプリケーションのライセンスへの抵触や著作権の侵害の可能性があるためである．

OpenCL においてカーネルコードを取得可能な API は幾つかあるが，本実装では clCreateKernel()において行った．ここでカーネル関数名を取得しておくことによって，使用リソース量の指定においてカーネル関数毎にリソース量の割り当てを行うことができ，また常駐サービスへの問い合わせ回数を減らすことができる．

(2) 実行時間の取得

プロセッサリソース割り当て機構はラッパーDLL であるため，API が呼び出されたタイミングでのみ時間の計測が可能である．そのため正確な実行時間の計測は不可能だが，clEnqueueNDRangeKernel()等のコマンドキューを利用する API が呼び出されたタイミングを開始時刻，キューの全てのタスクの終了を待つ clFinish()またはコマンドキューを解放する clReleaseCommandQueue()が呼ばれたタイミングを終了時刻として実行時間の計測を行う．

5.3.4 端末情報の取得・保存

4.3(4)の機能に対応し，デバイス及びリソースの配分決定の補助とするため，端末の CPU 使用率，電源の接続状態の情報を取得する．また，デバイス ID と実際のデバイスの対応付けのため，OpenCL の API を通じて取得できるデバイス情報の取得を行う．

(1) 端末情報の取得

常駐アプリケーションにおいて定期的に実行することで，使用デバイスおよびリソース量の決定の際に現在の CPU 使用率を取得できる．その他に必要な情報として，GPU 使用率等が考えられるが，本実装では取得しない．

(2) デバイス情報の取得

OpenCL では仕様上，デバイス ID は起動時に毎回異なる値を持つ．そのため，デバイス ID と実際のデバイスの対応付けを行う必要がある．デバイスの判別には clGetDeviceInfo()によって取得できるデバイス名を用いる．

本実装では，同一のデバイスが複数搭載されている端末ではデバイス名から同定が行えないため，デバイスを変更することができないという制限がある．しかし本研究では汎用携帯端末での利用を想定しているため，複数の同一デバイスが存在する構成を持つ場合は少ないと考えられる．

初めて常駐サービスを利用したときに一度だけ端末のデバイス情報を取得し，保存を行う．また，OpenCL の利用アプリケーションを実行して最初にラッパーDLL が呼ばれた際に，デバイス ID とデバイス名の対応表を取得しておき，デバイス ID とデバイス名の変換を行う．

6. 評価

6.1 動作検証

既存アプリケーションとの互換性の確認のため，複数のアプリケーションを用いて，動作検証を行う．ただし，アプリケーションは CPU･GPU 双方で利用可能なものである．

検証は以下の項目について行う．

- デバイスの切り替え可否
- リソース量の切り替え可否
- 実行情報及び端末情報の取得可否

動作検証には表 2 の環境を用いた．

検証結果は以下の通りである．

(1) OpenCL 入門 サンプルコード

文献[11]のサンプルコードのうち，4-3 dataParallel，6-1 fft，6-2 mt について検証を行った．

いずれも正常終了し，デバイス及びリソース量の切り替え，実行情報及び端末情報の取得が可能であった．

(2) APP SDK サンプルコード

AMD の APP SDK 2.7 With OpenCL 1.2 Support に同梱されている DCT プログラムで検証を行った．

正常終了し，デバイス及びリソース量の切り替え，実行情報及び端末情報の取得が可能であった．しかし，このサンプルは AMD APP SDK で用意されたライブラリを用いて実装されており，このライブラリでは処理の前準備として OpenCL が利用可能かの確認等の API 呼出を行うため，その際にも常駐サービスへの無駄な問い合わせが発生してしまい，処理の開始が遅くなった．

(3) GPU Computing SDK サンプルコード

NVIDIA の GPU Computing SDK[12]に用意されているサンプルコードである oclVectorAdd について検証を行った．

正常終了し，デバイス及びリソース量の切り替え，実行情報及び端末情報の取得が可能であった．

6.2 性能評価

提案方式の動画像通信における消費電力への影響を検証するため，本実装を用いてリソース調節を行った場合と行わない場合の消費電力量の比較を行う．

検証実験は以下の項目について行う．

表 2 動作検証の環境

製品名・型番	Magnate HB
CPU	Intel Core i5-3470
メモリ	PC3-12800 4096MB*2
チップセット	Intel B75 Express
グラフィックボード	NVIDIA GeForce GT 630
OS	Ubuntu 10.04 LTS 32bit

ⓒ 2013 Information Processing Society of Japan

表 3　消費電力の評価環境

製品名・型番	HP Folio 13-1009TU
CPU	Intel Core i5-2467M
メモリ	PC3-10600 4096MB*1
チップセット	Intel HM65 Express
OS	Ubuntu 10.04 LTS 32bit

- 本実装を用いず，使用デバイスに CPU を指定し，サンプルコードの初期値のリソース量を指定
- 本実装を用い，使用デバイスに GPU から CPU に変更し，サンプルのリソース量を変更

変更するリソース量については，予め評価環境における初期値より高速になる値を調査して利用する．

消費電力の計測は，簡易なものとしてモバイル PC を使用し Linux カーネルが提供している power supply class を利用する．したがって，計測値は端末全体の数値である．

検証には FFT を用い，512*512px の画像を処理する．一定の FPS（Frame Per Second）で画像を送信または受信することを想定し，FPS 毎の処理にかかる消費電力量を計測する．計測はバッテリを 100%まで充電した状態から 5 分間スリープさせたのち 10 分間処理を行い，処理中の消費電力量を計測する．検証では FPS に対して処理が間に合わず遅延が発生した場合，それ以上の計測を行わない．

性能評価には表 3 の環境を用いた．

消費電力の検証を行った結果が図 4 である．横軸が FPS，縦軸が実行中に計測した消費電力量の平均である．以降，提案手法を利用しない場合を通常と呼ぶ．

消費電力量は Linux カーネルが提供している power supply class の POWER_CURRENT_NOW の値を利用している． 全体として，通常に比べ提案方式の消費電力が低いことがわかる．また，FPS が小さい場合は処理量が少ないためリソース量を調節したことによる高速化効果が小さく，通常と提案方式に差が少ないが，FPS が大きいほど差が大きくなっており，より省電力効果が表れている．

また，通常の場合では FPS が 7 以下の値しか計測できていない．本検証では処理が FPS に対して遅延した場合，処理を停止するとしたため，これは 1 秒間に 7 フレームの処理が行えず遅延が起きたこと，つまり，低い送信レートでしか通信を行えないことを示している．提案方式は FPS が 12 まで計測できており，この差は通常に比べ提案方式の方が 1 回の処理時間が高速であること，より高い送信レートで通信が可能であることを示している．

6.3 考察

動作検証において，検証に用いた全てのサンプルにおいて本実装を利用してデバイス及びリソース量の切り替えが可能であった．(2)及び(3)は各 OpenCL フレームワーク実装のベンダが用意したライブラリを利用している．これらを用いて実装されたサンプルコードが本実装で動作することができたことから，多くのアプリケーションにおいて利用可能であると考えられる．

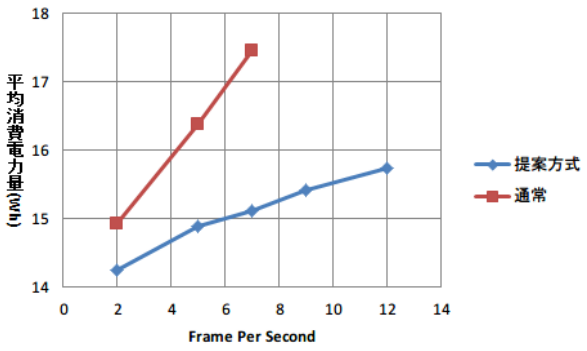

図 4　消費電力の比較

次に性能評価において，提案方式を用いた場合，処理が可能な FPS に約 1.7 倍の向上が見られた．実験で用いた FFT プログラムにおいてリソース調節を行った場合と行わなかった場合の実行時間比は約 1.57 倍であったため，期待された結果が出ている．リソース調節による処理の高速化により，提案方式を用いない場合に比べ，処理中の消費電力量は FPS が 2 のときでは約 4.56%，7 のときでは約 13.46%削減された．

7. まとめと今後の課題

本稿では，低消費電力で処理を行うために，汎用携帯端末においてプロセッサ抽象化 API を利用する際の互換性の課題を解決する方式を提案した．プロセッサの割り当てを決定する機能の分離を行うことにより汎用端末における互換性の解決を図った PACUE の手法を拡張して，汎用的な利用のために必要となる動的なリソース調節機能を追加し，汎用携帯端末における適用を目指した．提案方式を Linux OS 上に OpenCL を用いて実装し，汎用的に利用可能であるか，また動画像通信を例に消費電力の削減に効果があるかを検証した．

まず動作検証において，複数のサンプルコードでデバイス及びリソース量の変更ができた．次に消費電力の評価において，提案手法が動画像通信の低消費電力化及び高速化に有効であることがわかった．

今後は，Android において実装を行い，汎用携帯端末上における効果の検証を行う．また，今回の性能評価は CPU のみを用いてリソース調節を行った場合であったが，提案方式は動的なプロセッサの割り当てが可能であるため，実験でも行った画像処理のような GPU が高速化に有効なアプリケーションでは GPU を活用するとより効果が見込める．汎用携帯端末の GPU でも OpenCL を利用できる環境が整いつつあるため，GPU を利用し，負荷分散を行った場合の性能評価も行いたい．また，マルチスレッドにおける動作検証，異なる種類のデバイス及びフレームワークにおけ

ⓒ 2013 Information Processing Society of Japan

る実験，プロセッサ及びリソースの配分を決定するアルゴリズムの検討を行いたいと考えている．

参考文献

1) OpenCL, http://www.khronos.org/opencl/
2) 朴泰祐，佐藤三久，塙敏博，児玉祐悦，高橋大介，建部修見，多田野寛人，藏増嘉伸，吉川耕司，庄司光男：演算加速装置に基づく超並列クラスタ HA-PACS による大規模計算科学, 情報処理学会研究報告，Vol.2011-HPC-130，No.21，pp.1-7(2011)
3) Tetsuro Horikawa, Michio Honda, Jin Nakazawa, Kazunori Takashio, Hideyuki Tokuda : PACUE: Efficient Heterogeneous Processor Allocation in PCs, Workshop on UnConventional High Performance Computing 2011, Euro-Par 2011, Vol.7156, pp.335-344(2012)
4) Hadi Esmaeilzadeh, Emily Blem, Renee St.Amant, Karthikeyan Sankaralingam, Doug Burger : Dark silicon and the end of multicore scaling, Proceedings of the 38th annual international symposium on Computer architecture, ISCA'11, pp.365-376(2011)
5) Hiroyuki Takizawa, Katsuto Sato, Hiroaki Kobayashi : Sprat : Runtime proessor selection for energy-aware computing, Proceddings of the 2008 IEEE International Conference on Cluster Computing, pp.386-393(2008)
6) Kyle Spafford, Jeremy Meredith, Jeffrey Vetter : Maestro: data orchestration and tuning for oppencl devices, Preceeding of the 16th international Euro-Par conference on Parallel processing: PartⅡ, Euro-Par'10, pp.275-286(2010)
7) Victor J Jimenez, Lluis Vilanova, Isaac Gelado, Marisa Gil, Grigori Fursin, Nacho Navarro, Predictive Runtime Code Scheduling for Heterogeneous Architectures, Proceedings of the 4th International Conference on High Performance Embedded Architectures and Compilers, HiPEAC'09, pp.19-33(2009)
8) 島田大地，遠藤敏夫，丸山直也，松岡聡：OpenCL を用いた異種 GPU における性能特性に応じた最適化, 情報処理学会研究報告, Vol.2010-HPC-128，No.23，pp.1-7(2010)
9) AMD, Accelerated Parallel Processing (APP) SDK,http://developer.amd.com/tools/heterogeneous-computing/amd-accelerated-parallel-processing-app-sdk/
10) NVIDIA, CUDA http://www.nvidia.com/object/cuda_home_new.html
11) 株式会社フィックスターズ, 土山了士, 中村孝史, 飯塚拓郎, 浅原明広，三木聡：OpenCL 入門 -マルチコア CPU・GPU のための並列プログラミング-，インプレスジャパン(2010)
12) NVIDIA, GPU Computing SDK, https://developer.nvidia.com/gpu-computing-sdk

ⓒ 2013 Information Processing Society of Japan